
1 Host-Security

In diesem Kapitel lernen Sie

- ▶ die Gefahren kennen, denen ein Linux-System durch unerwünschtes Fehlverhalten einiger User ausgesetzt sein kann, und wie man sich dagegen wappnet.
- ▶ die verbreitetsten Angriffe kennen.

Jede Software enthält Fehler ¹.

Softwarefehler sind zum Teil einfach nur ärgerlich, zum Teil können sie aber auch kritisch für das Überleben eines Unternehmens werden.

Insbesondere solche Fehler, die es *Menschen* ermöglichen, an

- Rechte

oder

- Informationen

zu gelangen, auf die sie normalerweise keinen Zugriff haben sollten, sind im Unternehmensumfeld extrem kritisch einzustufen.

Da dieses „Naturgesetz“ besteht, sollte man sich als Administrator zumindest mit den populären Angriffstechniken auseinandersetzen.

UNIX-Systeme mit ihrem vergleichsweise flach gestaffelten Rechtesystem (`root` darf alles!) sind relativ beliebte Ziele sowohl für Netzwerk- als eben auch für lokale Angriffe, die sich vor allem gegen den `root`-Account richten:

Hat ein User erst einmal diesen Account geknackt, steht ihm das gesamte System offen, er kann sogar problemlos den „echten“ `root` aussperren.

Dennoch sind für bestimmte Aufgaben im System `root`-Rechte zwingend nötig:

- Prozesse, die direkten Zugriff auf die Hardware benötigen (`X-Server`, `mount`, ...)
- Manche Programme müssen „im Userauftrag“ Dateien manipulieren, die nur für `root` schreibbar sind (`passwd`, `crontab`, ...)

¹Nicht ganz: Vom Satzsystem `TEX` sind nach über 20 Jahren nur 3 Fehler bekannt, und diese sind in 90% der Fälle erwünscht ;-). Der Autor, Professor Knuth, verspricht, jedem Finder eines Bugs einen beträchtlichen Betrag zu zahlen...

- Prozesse, die sich an privilegierte (Portnummer kleiner als 1024) Netzwerkports binden sollen, benötigen dafür zumindest zeitweise `root`-Rechte (**httpd**, **nfsd**, **portmap**, . . . , aber z.B. auch **rsh**, das von Usern ausgeführt werden kann)

Diese Prozesse werden zum Teil *SUID-root* installiert (**rsh**, **mount**, . . .), zum Teil werden sie bereits beim Boot-Vorgang oder später vom `root` gestartet.

Solche Prozesse bzw. Programme sind begehrte Ziele von Angriffen, da ein aufgespürter Fehler gegebenenfalls ohne weitere Umwege einen direkten Zugriff auf den `root`-Account ermöglicht.

Zwei typische Angriffsformen sollen hier erörtert werden, der sog. *buffer overflow* sowie die *tmp-race*-Problematik.

1.1 Buffer Overflow

Die häufigste Sicherheitslücke unter Unix sind fehlerhafte Programme, die ihre Eingaben nicht auf ihre Korrektheit (Länge oder Sonderzeichen) prüfen, was zu dem als *buffer overflow* bekannten Problem führt:


Ein Prozess erhält eine Usereingabe, der Programmierer sieht für diese Eingabe einen Speicherbereich vor (den *buffer*). Ist der Programmierer jetzt der Meinung, dass z.B. 40 Buchstaben für einen Nachnamen reichen, reserviert er beim Betriebssystem für die Speicherung dieses Namens einen Bereich von 41 Bytes (40 Buchstaben und der C-typische String-Terminator, die binäre Null).

Wird die Usereingabe nicht ausreichend auf ihre tatsächliche Länge überprüft, und gibt ein User beispielsweise programmgesteuert statt der vorgesehenen maximal 40 Zeichen 2000 ein, kann er damit andere Speicherbereiche überschreiben. (Der *buffer* läuft über: *buffer overflow*)

Diese überlange Eingabe wiederum kann, wenn die Zeichen 41–2000 geschickt gewählt sind, den Prozess dazu veranlassen, (im besten Falle) abzustürzen oder dem User eine Shell mit den Rechten des Prozesseigentümers zu öffnen.

Kritisch ist das vor allem bei den oben beschriebenen *SUID-root*-Prozessen.

So kann ein schlechtes Passwort einem Angreifer einen Remote-Zugang als normaler User per **telnet** öffnen, den er dann mit einem fehlerhaften *SUID-root*-Programm zu einem Administrator Zugang „ausbauen“ kann.

Hinweis: Eine Variante dieser Problematik sind Angriffe auf selbstgeschriebene Skripte eines unerfahrenen oder unvorsichtigen `root`: 

Weiß ein User, dass der `root` täglich ein kleines „Cleaner“-Skript laufen lässt, das `/tmp` mit dieser Kommandozeile aufräumen soll:

```
# find /tmp -mtime +3 -exec rm -f {} \;
```

so könnte er in Versuchung geraten, den **vi** als `/tmp/copy_of_vi` zu kopieren, sowie einen Tag vorher eine Datei namens

```
*root.root*; chown root.root copy_of_vi; chmod u+s copy_of_vi anzulegen.
```

Diese Datei würde dem User für einen Tag eine *SUID-root*-Kopie des **vi**-Editors zur Verfügung stellen, aus dem heraus er per **:sh** sofort eine *root*-Shell aufrufen könnte.

Das Perfide an einem solchen Angriff wäre, dass das einzige Beweismittel (die Datei in `/tmp`) durch den vorangegangenen Aufruf von **rm -f /tmp/*root.root*** gelöscht worden wäre².

1.2 Die *tmp-race*-Problematik

Ein weiterer „populärer“ Programmierfehler ist die fehlerhafte Benutzung temporärer Dateien:

Öffnet z.B. ein Programm wie **crontab** eine Datei in `/tmp`, um die geschützt abgelegte Datei `/var/spool/cron/username` dort hineinzukopieren, *ohne* vorher zu überprüfen, ob die temporäre Datei vielleicht schon existiert, stellt dies eine erhebliche Sicherheitslücke dar:

Gelingt es einem Angreifer, den Namen der temporären Datei im Vorfeld zu erraten (z.B. `/tmp/crontab.meine PID+5`), kann er einen symbolischen Link auf z.B. `/etc/passwd` anlegen.

Der Kopiervorgang würde dann (**crontab** ist *SUID-root* installiert) die `/etc/passwd` überschreiben (die Betriebssystemfunktion `open()` folgt Symlinks)!

Die Folge wäre ein *DOS Denial Of Service*: der Rechner erlaubt keine Anmeldung mehr, da die Benutzerdatenbank überschrieben wurde.

1.3 Gegenmaßnahmen

Gegen solche und ähnlich gelagerte Sicherheitslücken hilft vor allem

- die Deinstallation bzw. Entschärfung (**chmod u-s file**) aller nicht zwingend benötigten *SUID-root*-Programme (so sind z.B. zahlreiche **Gnome**-Spiele

²Ganz so einfach ist es aber zum Glück nicht, **find** durcheinanderzubringen. Das Beispiel soll nur als Demonstration dienen!

Sie brauchen dies also nicht auszuprobieren. . . .

SUID-root installiert, damit sie Spielstände in `/usr/share/...` abspeichern können!)

- das sofortige Beheben aktueller Sicherheitslücken durch Einspielen der entsprechenden Patches (Open-Source-Software bietet hier den Vorteil, dass die Problem-Erkennung meist mit der -Behebung einhergeht)
- der Einsatz von als zuverlässig bekannten Tools statt eigener Skripte, wo immer es geht (so nimmt einem z.B. **tmpwatch** das Nachdenken über die oben geschilderte Problematik ab)
- das Abschalten aller nicht zwingend benötigten Netzwerk-Dämonen (s.u.)

Um dabei auf dem Laufenden zu bleiben, empfiehlt sich ein Abonnement der Mailingliste *bugtraq* oder der entsprechenden Liste des Distributors (z.B. *suse-security-announcements*).

1.3.1 *SUID*-Programme entschärfen

Die Suche nach Programmen mit gesetztem *suid*-Bit mittels

```
# find / -perm +4000 -type f
```

bringt eine erstaunlich lange Liste hervor, so u.a.:

/bin/su Wechselt die User-ID (switch user)

/bin/ping Direkter Zugriff auf die unterste Netzwerkebene (data-link-layer)

/bin/eject Direkter Zugriff auf den CD-ROM-Treiber

/bin/mount Damit ein normaler Benutzer CD-ROMs und Disketten mounten kann

/usr/bin/man Damit ein Benutzer den Index in `/var/cache/man` speichern darf

/usr/bin/lpr Zum Ablegen der Druckjobs in `/var/spool/lpd`

/usr/bin/rsh Zum Öffnen eines privilegierten Client Ports

/usr/bin/chsh Zum Ändern der `/etc/passwd`

/usr/bin/crontab Zum Ablegen von **cron**-Jobs in `/var/spool/cron/`

Alle *SUID* und *SGID*-Bits können mit folgendem Befehl gelöscht werden:

```
# find / -perm +4000 -o -perm +2000 -type f | xargs chmod ug-s
```

Damit wird die Nutzbarkeit für normale User stark eingeschränkt, bei einer Firewall oder einem dedizierten Server ist das aber akzeptabel, da sowieso keine lokalen User existieren (sollten).

1.3.2 Sonderfall Dateirechte bei der SuSE-Distribution

Das Skript **SuSEconfig** setzt die Dateirechte nach den Einträgen in den Dateien `/etc/permissions*` wieder zurück. Dabei wird zuerst die Konfigurationsdatei `/etc/sysconfig/security` eingelesen und dann die in der Variablen `PERMISSION_SECURITY` angegebenen Skripte gestartet.

Die Voreinstellung sieht wie folgt aus:

```
PERMISSION_SECURITY="easy local"
```

Statt der `/etc/permissions.easy` existiert auch die `permissions.secure` und die `permissions.paranoid`.

Alternativ können in der sonst leeren Datei `/etc/permissions.local` alle Dateirechte erfasst, korrigiert und dann mit **SuSEconfig** gesetzt werden. Das Eintragen lässt sich wie folgt automatisieren:

```
# find / -perm +4000 -o -perm +2000 \  
> -printf "%h/%f %u.%g %m\n" >/etc/permissions.local
```

Natürlich lässt sich **SuSEconfig** auch davon abhalten, die Rechte zu ändern. Dazu wird die Variable `CHECK_PERMISSIONS` auf `no` gesetzt.

Beide Variablen werden mit dem Sysconfig-Editor von YaST2 (System→Editor für `/etc/sysconfig`-Dateien) geändert. Sie finden sich dort unter System→Security→Permissions.



Hinweis: In älteren SuSE-Versionen (vor SuSE 8) stehen diese Variablen noch in der Konfigurationsdatei `/etc/rc.config`

Dabei sollte man sich nicht in der falschen Sicherheit wiegen, der eigene Rechner sei zu unwichtig/unbekannt oder durch seine ständig wechselnde, dynamische IP-Adresse ein schlechtes Ziel.

Viele Angriffsprogramme testen vollautomatisch und permanent große IP-Adressbereiche auf bekannte Sicherheitslücken, so dass wirklich jeder Rechner im Internet einer ständigen Gefahr ausgesetzt ist – vor allem ältere, nicht gepatchte Betriebssystemversionen, Linux wie Windows, sind ein leichtes Ziel.³

³So sagt man (Sept. 2001), dass ein RedHat 6.1-System (Release Frühjahr 2000) ohne jeden Sicherheitspatch durchschnittlich 15 Minuten (!) nach dem Anschluss an das Internet geknackt ist.

1.4 Dämonen

Bei einer Standard-Linuxinstallation sind bereits etliche Dienste (Netzwerk und lokal) aktiv.

Die wenigsten davon stellen gewünschte Dienste dar oder sind für den Betrieb notwendig und bei einer Firewall sollten alle Netzwerkdienste abgeschaltet werden.

Auch sonst sollte man sich auf das unbedingt Benötigte beschränken.

Es gibt zwei Möglichkeiten, Serverdienste anzubieten:

1. Aufruf während des Boot-Vorgangs über ein Startskript (oder manueller, nachträglicher Start durch den Systemverwalter)
2. der indirekte Start bei Bedarf über den **inetd** bzw. **xinetd**

1.4.1 Der [x]inetd

Der **inetd** ist der *Internet Super-Server*. „Super“ deshalb, weil er *über* den durch ihn verwalteten Netzwerkdiensten steht; er überwacht mehrere Netzwerkports und startet bei Bedarf (eingehende Verbindung) den zum jeweiligen Port gehörenden Server-Prozess.

Der **[x]inetd** ist die *eXtended* Version des **inetd**.

Die Erweiterungen schließen Dinge wie eingebaute, host-basierte Zugangskontrolle oder z.B. die Einschränkung von Diensten auf bestimmte Tageszeiten ein.

Über den **inetd** angebotene Dienste lassen sich sehr schnell mit **grep** auffinden:

```
grep -v "^#" /etc/inetd.conf
```

Bei einer SuSE 7.0 Standardinstallation sind z.B. folgende Dienste eingetragen:

```
daytime stream tcp      nowait root    internal
daytime stream udp      nowait root    internal
time     stream tcp      nowait root    internal
time     dgram  udp      wait   root    internal
ftp      stream tcp      nowait root    /usr/sbin/tcpd  in.ftpd -a
telnet   stream tcp      nowait root    /usr/sbin/tcpd  in.telnetd
nntp     stream tcp      nowait news   /usr/sbin/tcpd  /usr/sbin/leafnode
shell    stream tcp      nowait root    /usr/sbin/tcpd  in.rshd -L
login    stream tcp      nowait root    /usr/sbin/tcpd  in.rlogind
talk     dgram  udp      wait   root    /usr/sbin/tcpd  in.talkd
ntalk    dgram  udp      wait   root    /usr/sbin/tcpd  in.talkd
pop3     stream tcp      nowait root    /usr/sbin/tcpd  /usr/sbin/popper -s
finger   stream tcp      nowait nobody  /usr/sbin/tcpd  in.fingerd -w
```

Dazu sei angemerkt:

time, daytime: Die Zeitserver sind vermutlich sicher, weil sie keinerlei Eingabe erwarten und keine externen Programme benötigen.

telnet, ftp, shell, login: Hier werden Passwörter im Klartext übertragen; außerdem hatten etliche dieser Server in der Vergangenheit mit Sicherheitslücken zu kämpfen.

Wenn möglich, sollten stattdessen die sicheren Alternativen **ssh**, **scp** und **sftp** verwendet werden! Es gibt heutzutage wirklich keinen Grund mehr, telnet statt SSH einzusetzen.

finger: Dieser Dienst verrät auch bei normaler Funktion etliche sensible Informationen über die angelegten Benutzer und das Betriebssystem. Er sollte daher auf auf jeden Fall abgeschaltet werden.

talk, ntalk: Damit kann ein Angreifer eine Chat-Verbindung zu einem Benutzer im internen Netzwerk aufbauen. Da unbedarfte Anwender oftmals die größte Sicherheitslücke darstellen, sollte auch dieser Dienst deaktiviert werden.

pop3: Das *pop3*-Protokoll überträgt Passwörter im Klartext.

Der Dienst sollte also nur aktiviert sein, wenn der Host tatsächlich ein *pop3*-Server ist.

nntp Auch hier gilt: ist dieser Host nicht der News-Server des Unternehmens, kann und sollte der Dienst abgeschaltet werden.

Ein Dienst wird aus der Verwaltung des **inetd** herausgenommen, indem in der `/etc/inetd.conf` am Beginn der entsprechenden Zeile eine Raute (#) eingetragen wird. Anschließend muss dem Dienst noch das Signal `-SIGHUP` geschickt werden (`# killall -SIGHUP inetd` oder einfach `# /etc/init.d/inetd restart`).

Wenn Sie auf alle Dienste, die unter der Verwaltung des **inetd** stehen, verzichten können, kann dieser natürlich auch ganz abgeschaltet werden (s.u.).

Die Konfiguration des **[x] inetd** findet über die Datei `/etc/xinetd.conf` statt.

Diese unterscheidet sich syntaktisch erheblich von der `inetd.conf`, ist aber im direkten Vergleich doch sehr ähnlich. Hier haben die einzelnen Dienste eigene Abschnitte:


```
service telnet
{
    disable = yes
    flags          = REUSE
```

Host-Security

```
socket_type    = stream
wait           = no
user           = root
server         = /usr/sbin/in.telnetd
log_on_failure += USERID
}
```

Im abgedruckten Beispiel sieht man auch sofort, wie ein Dienst im **[x]inetd** abgeschaltet wird:

Im Abschnitt des Dienstes wird die Zeile „`disable = yes`“ eingetragen.

Hinweis: Die **[x]inetd**-Konfiguration wird z.B. von RedHat (ab 7.0) über eine `include ...`-Anweisung auf verschiedene Dateien in `/etc/xinetd.d` verteilt. Gleichzeitig unterliegt sie in diesem Fall aber auch der Kontrolle von **ntsysv** und **chkconfig** (s.u.)! 

Dem **[x]inetd** sollte man besser kein Signal `-SIGHUP` schicken (veranlasst einen *state dump* und lässt den Admin in trügerischer Sicherheit...). Er wird besser gleich über das *init-Skript* neu gestartet/geladen (`# /etc/init.d/xinetd restart` bzw. `reload`).

1.4.2 Startskripte

Die sog. *standalone-Server* wie z.B. *Samba* (`smbd & nmbd`), *Apache* (`httpd`), *BIND* (`named`) aber auch der **[x]inetd** werden durch eigene *init-Skripte* gestartet.

Um einen solchen Dienst zu deaktivieren, entfernt man den symbolischen Link `/etc/rcX.d/SnnDienst`, wobei *X* der zu editierende Runlevel (Betriebszustand, meist 3) und *nn* eine Zahl zwischen 00 und 99 ist.

Die meisten Distributionen bringen aber auch Tools mit, die diese Aufgabe übernehmen.

Bei SuSE mit dem folgenden Befehl (Im Beispiel soll Apache in den Runleveln 2, 3 und 5 gestartet werden):

```
# chkconfig apache 235
```

Man kann aber auch den Runlevel-Editor von SuSE verwenden. Der findet sich bei YaST unter System→Runlevel-Editor.

Bei Fedora/RedHat lautet der Befehl (Im Beispiel soll Apache in den Runleveln 2, 3 und 5 gestartet werden):

```
# chkconfig --level 235 httpd on
```

Als grafisches Administrations-Tool für diese Aufgabe gibt es bei Fedora/RedHat das Programm **system-config-services**.

Bei Debian lautet der Befehl, um Apache in den Runleveln 2, 3 und 5 zu starten und in den Runleveln 0, 1 und 6 zu stoppen:

```
# update-rc.d apache2 defaults
```

Alternativ zur manuellen Überprüfung seiner Konfiguration kann man sich auch die Verbindungen im Zustand `LISTEN` anzeigen lassen, um alle noch aktiven Serverdienste aufzuspüren:

```
# netstat -nlutp
```

Die Optionen haben dabei folgenden Bedeutung:

- n Keine Namensauflösung
- l Nur Verbindungen im Zustand `LISTEN`
- ut UDP- und TCP-Verbindungen, keine Unix Domain Sockets
- p Zeigt auch die dazugehörigen Prozesse

1.5 Physikalischer Zugriff

Hat ein Angreifer direkten Zugriff auf die Hardware, ist meistens schon alles verloren, denn es gilt die Grundregel:

Physikalischer Zugriff ist gleich Vollzugriff

Trotzdem sollte man es ihm nicht unnötig leicht machen (sofern der Standort des Rechners ein öffentlicher oder videoüberwachter Raum ist, ist das Aufschrauben eines Rechners nicht gerade unauffällig).

Es gibt aber auch ein paar Punkte, die bei einem Arbeitsplatzrechner zu beachten sind; beispielhaft sei hier auf ein Problem mit dem Bootloader *LILO* hingewiesen.

1.5.1 Sicherheitsloch Bootmanager

Ein beliebter Angriffspunkt für Angreifer mit physikalischem Zugriff ist die Parameterübergabe des Bootloaders *LILO* (*L*inux *L*Oader) an den Kernel:

- Der LILO untersucht die ihm am Prompt übergebene Zeile nach ihm bekannten Namen von Boot-Images (z.B. `linux`)
- Ist das ausgewählte Image eine Linux-Installation, wird die Zeile weiter durchsucht, und zwar nach Parametern für LILO (`vga=ask` etc. pp.)
- Was dann noch von der Zeile überbleibt, wird dem Kernel übergeben
- Dieser wiederum untersucht die Zeile nach ihm bekannten Parametern (z.B. `eth0='ne2000,irq=10,io=0x300'`)
- Was nach dieser Analyse noch immer unerkannt geblieben ist, wird dem **init**-Prozess übergeben

Diese Vorgehensweise hat durchaus ihre Existenzberechtigung, reißt aber auch ein Sicherheitsloch auf:

Gebe ich z.B. am LILO-Prompt als zu bootendes Image `linux 1` an, startet das System im *Single-User-Modus* (weder LILO noch der Kernel können mit `1` etwas anfangen, **init** aber sehr wohl; probieren Sie einmal `# init 1!`).

Bei SuSE ist dies nicht so verheerend, da im Runlevel 1 ein **sulogin** gestartet wird, welches nach dem `root`-Passwort fragt.

Fedora/RedHat sieht hingegen den Runlevel 1 explizit für den Fall vor, dass z.B. durch einen Angriff (wie er oben geschildert wurde) oder durch Misskonfiguration ein ordentliches Login als `root` nicht mehr möglich ist. Daher wird ohne weiteres Zutun eine Shell mit `root`-Rechten gestartet.

Doch auch die **sulogin**-Konfiguration hilft nicht wirklich:

Der Kernel versteht eine Option `init=Datei`, die den Dateinamen der ausführbaren Datei für den Init-Prozess (standardmäßig ist das natürlich `/sbin/init`) überschreibt.

Gibt man am LILO-Prompt als **init**-Programm eine Shell seiner Wahl an (z.B. `per linux init=/bin/bash`), startet diese ohne Passwortabfrage.

Um das zu verhindern, gibt es die Möglichkeit, in der `/etc/lilo.conf` bei den globalen Optionen einen Eintrag

```
restricted
```

vorzunehmen, der noch durch eine weitere Zeile mit dem Passwort

```
password=...
```

ergänzt werden muss.

Diese beiden Einträge bewirken, dass bei jeder „unnormalen“ Eingabe am LILO-Prompt nach dem Passwort gefragt wird.